Towards adaptive autosar : a system-level approach

Amel Belaggoun, Valerie Issarny

F2016-VESE-003

TOWARDS ADAPTIVE AUTOSAR: A SYSTEM-LEVEL APPROACH

[1]BELAGGOUN,Amel[*]; [2]ISSARNY,Valerie; [1]RADERMACHER,Ansgar
[1]CEA LIST, Laboratory of Model Driven Engineering for Embedded Systems, B.P.174-91 Gif-sur-Yvette, France; [2]INRIA-Paris Rocquencourt, France

KEYWORDS– AUTOSAR, runtime adaptation,mixed hard/softreal-time embedded systems, dynamic reconfiguration,task allocation.

ABSTRACT

The automotive industry has recently agreed upon the embedded software standard AUTOSAR, whichstructures an application into reusable components that can be deployed according to a given configuration. However, this configuration is fixed at design-time, with no support for dynamically adding, migrating and/or removing components to adapt the system at runtime. In this paper, we present the design and implementation of ASLA (Adaptive System-Level in AUTOSAR) a framework that provides runtime adaptation to AUTOSAR. We believe that our approach opens up new functionalities for vehiclesoftware platforms and can be leveraged in therecent initiative Adaptive AUTOSAR.

INTRODUCTION

More than ten years have passed since the establishment of the AUTOSAR(Automotive Open System Architecture) standard. AUTOSAR was introduced to simplify automotive system design while offering interoperability, scalability, and extensibility. Its current status does not support runtime adaptation; i.e.: changing the system's structure and/or behavior at runtime in response to environmental changes or failures. In the standard, the system configuration is static by design from the application down to the Operating System (OS) layer. A reconfiguration of the system, such as adding an application or moving an application from one Electronic Control Unit (ECU) to another cannot be done dynamically at runtime. Making AUTOSAR adaptive requires specific support at different layers of the software architecture. Therefore, our objective is to elaborate an architectural solution that can handle the adaptation of mixed criticality applications while respecting timing and safety requirements and offering a high degree of flexibility. To address this challenge we have developed a layer called ASLA (Adaptive System-Level in AUTOSAR) to incorporate task-level adaptation techniques in AUTOSAR. The key contribution of our paper is to describe the implementation-driven design of ASLA.

The rest of the paper is organized as follows. The next sectionintroduces AUTOSAR central concepts.Then, we describe the main contribution of this paper, namely the design and the implementation of ASLA. After that, ASLA is evaluated on an AUTOSAR-compliant adaptive platform implementation.Finally, we provide our concluding remarks and the lessons learned in the final section.

AUTOSAR OVERVIEW

Fig.1 depicts the architecture of an AUTOSAR compliant ECU. Briefly stated, AUTOSAR [1] is a layered software architecture that decouples application software (see Fig.1-1) from the lower level *Basic SoftWare*(BSW) (see Fig.1-3). The BSW consists of an OS that has evolved from the OSEK standard [2]; system services for, e.g., memory management; communication concepts; ECU and microcontroller hardware abstractions; and complex device drivers for direct access to hardware.

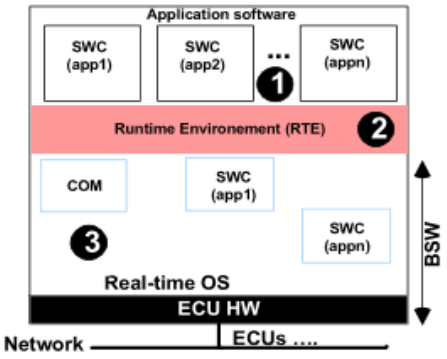

Figure 1: A simplified AUTOSAR ECU Architecture

The application software consists of set of *Software Components*(SWCs). Each SWC has a number of ports for communication with the rest of the system (they can be either required or provided ports). The internal functionality, or the*runnable*, of the component only accesses its ports, and not directly any other components, which promotes reuse and transferability of SWCs across ECUs. The runnables are allocated to OS tasks. The communication between SWCs, as well as between SWCs and the underlying software layers, is based on a concept called *Virtual Function Bus* (VFB). The idea of VFB is to allow SWCs to communicate with each other as if they were all allocated to the same ECU. If they are in fact on different ECUs in a particular implementation, the communication between them has to be mapped to network messages without involving the SWCs themselves. The*Runtime Environment* (RTE) (see Fig.1-2) is the realization ofVFB, providing the actual message transfer to and from SWC ports. The RTE also provides an API to the application software level, and in turn calls the API of the BSW. Apart from communication, the RTE also handles other functionalities, such as events, critical sections, etc. In addition to the technicalconcepts of AUTOSAR standards, AUTOSAR provides a development methodology [3], which relies on different tools for software configuration, including BSW composition, allocation of SWCs to ECUs, and dependencies between SWC ports, which are defined statically at design time in a number of description files. These files are then processed by AUTOSAR tools, creating executable software that implement the BSW, RTE, and the application software for a particular ECU.AlthoughAUTOSAR provides a lot of flexibility in reconfiguring a system, it has the following shortcomings:

- AUTOSAR[1] adopts a static configuration approach where the whole system configuration with all its resources is known at design-time.
- The AUTOSAR RTE [4] is configured at design-time for specific ECUs and partly generated based on the requirements of the SWC. A reconfiguration of the system, such as adding an application or moving an application from one ECU to another, cannot be done dynamically at runtime. In addition, the AUTOSAR OS supports a fixed priority scheduling mechanism that assigns static priority to tasks, and hence, does not offer any possibility to create new tasks at runtime.

These Shortcomings of AUTOSAR for automotive systems motivate the ASLA solution.

ASLA'S SYSTEM AND FAILURE MODELS

ASLA supports applications with mixed hard/soft real-time requirements [5]. We assume that the system comprises *n* ECUs communicating via messages over a Control Area Network (CAN) [6], where each ECU has a processor executing real-time periodic hard/soft tasks/runnables τ_i that share a common memory. Hard tasks/runnablesare represented by (C_i , T_i, D_i), where each task τ_i computes for a maximum of C_i time-units every T_i time-units within a deadline D_i. Soft tasks/runnables are represented by the probability distribution functions(*PDFs)* $U_i$of their execution times and a soft deadline $\delta_i(PDFs\ U_i, \delta_i)$[5].The soft tasks/runnablesare scheduled usingCBS (Constant bandwidth server)[7].Thus, each soft task/runnable is assigned a CBS, characterized by the tuple (Q_i,T_i), where Q_i (also called bandwidth) is the time that soft task/runnable is allowed to use the CPU every periodT_i.The hard tasks/runnables andCBS servers are scheduled under EDF(Earliest Deadline first)[8] policy. CBS enforces temporalisolation between hard and soft real-time tasks, thus guaranteeingthe schedulability of hard tasks. In this work all tasks/runnables are periodic with implicit deadline i.e., $D_i = T_i$. This choice relies on the current state of industrial applications in the automotive field which are often of periodic nature.

This paper focuses on a fail-stop model of failures, where tasks or ECUs can fail and the remainder of the system can continue executing. In other words, tasks running on a live ECU are assumed to always emit correct outputs. Therefore, in order for the system to continue to correctly operate, we assume that ASLA employs a failure recovery policy [9], which consists of restoring the last non-faulty state of the failing task, i.e., to recover from faults. This state has to be saved in advance in the shared memory and will be restored if the task fails. We assume that the network provide bound bounded communication latencies and do not fail. This assumption is reasonable for automotive systems. Relaxing this assumption through the integration of our framework with the network level fault-tolerance techniques is an area of future work.

DESIGN AND IMPLEMENTATION OF ASLA

We now describe how ASLA is designed to overcome the limitations of AUTOSAR. We first detailthe responsibilities of the major components of ASLA. Then we describe how these components work together with different runtime algorithms to provide tasks adaptation capabilities in AUTOSAR.

ASLA'SARCHITECTURE

Fig.2 provides an overview of the ASLA architecture. Every ECU that supports adaptation through the ASLA layer consists of a real-time OS with EDF and CBS scheduling policies, an Adaptive SWC that is responsible for reconfiguringapplications running on the system, RTE, and application layers. The real-time OS is

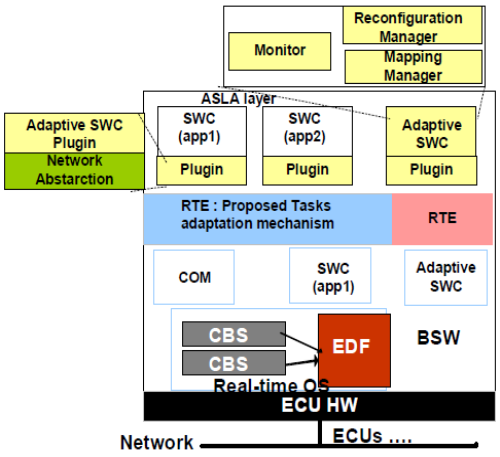

Figure 2 : The ASLA architecture

responsible for HW abstraction, communication, scheduling and executing tasks in real-time. We assume that the underlying HW is a fail-silent system and the communication network is fault-tolerant. Our application layer consists of a set of SWCs (similar to AUTOSAR's) and a new Adaptive SWC which can be distributed over several ECUs. The RTE provides a communication abstraction to SWCs. Unlike AUTOSAR, our RTE extension contains functions to support adaptation. These functions are managed by the Adaptive SWC (more precisely by the Reconfiguration Manager (RM)) which also communicates with the other Adaptive SWCs running on the different ECUs to make one of the adaptation actions such as: adding, deleting or updating application. When a new application is being added, the mapping between the application's SWCs and the ECUs is given to the RM, and then each RM analyzes the mapping and renews the RTE's function. The ASLA layer is composed of an Adaptive SWC (one on each ECU) and plugin offering a task execution container. This plugin enables any task launched on the ASLA layer to be periodically executed. The adaptive component adheres to a coordination-based architecture. One Adaptive SWC acts as a coordinator of the other Adaptive SWCs which are responsible for handling tasks on each ECU and monitoring a health vector. The latter contains all Non-Functional Requirements (NFRs) needed for the adaptation such as the ECU's processor utilization, resources, QoS, HW NFRs...etc. All operational ECUs compute their resources and processor utilization in form of a health vector at a fixed time period and share their health vector with each other. This provides each ECU a consistent view of the available resources and utilization on the other nodes,Fig.3. illustrates a brief overview of our health vector.

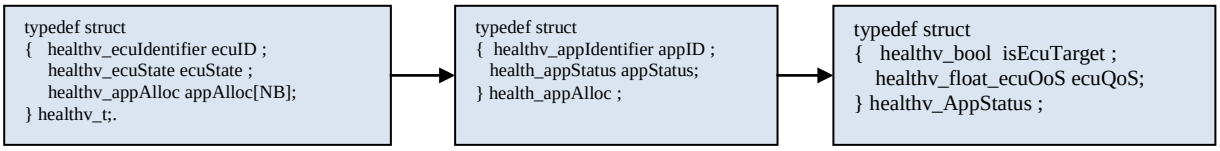

Figure 3: An example of health vector structure

Since our Adaptive SWC follows a coordination based-architecture, we define a management protocol between the different Adaptive SWCs inspired by [10]. However, we differ from them in the sense that our protocol is used for managing the process of an adaptation in distributed real-time systems. In our protocol, all Adaptive SWCs including the coordinator broadcast messages to each other. The coordinator can detect the failure of the others Adaptive SWCs by the lack of heartbeat messages. The major components of ASLA are described below.

The Adaptive SWC:As illustrated in Fig.2,the Adaptive SWC is composed of a monitor, a Mapping Manager (MM) and a Reconfiguration Manager(RM). The monitor is responsible for monitoring events that trigger the adaptation. The MM offers a dynamic deployment of tasks on the ECUs and the RM can automatically reconfigure tasks inside/or between the different ECUs:

- The Monitor. The monitor periodically sends messages to other ECUs in the system via the network.The monitor allows ASLA to agree on the availability of each ECU. Any adaptation trigger received by the application during its execution may invoke the monitor which sends a message to the RM in order to adapt the application. The loss of a message for two consecutive cycles means that the ECU is no longer alive and the adaptation needs to be triggered to accommodate the desired changes.

- The Mapping Manager.The MM offers an automatic deployment of tasks on ECUs. We use the O-TSMBA (Operational chains-TSMBA) algorithm[1], a variant of TSMBA[5] that supports task dependencies. The MM takes as input the application description (an initial system configuration file) and changes the current mapping when it is necessary to do so. Changes of the allocation can occur due to the adaptation or in case of one or several ECUs failures.

- The Reconfiguration Manager. The RM is a sporadic task that gets triggered upon the reception of an adaptation trigger (requests for adding new tasks, requests for migrating failed tasks/and or failed ECUs, replacement of tasks with an improved version and removing tasks).

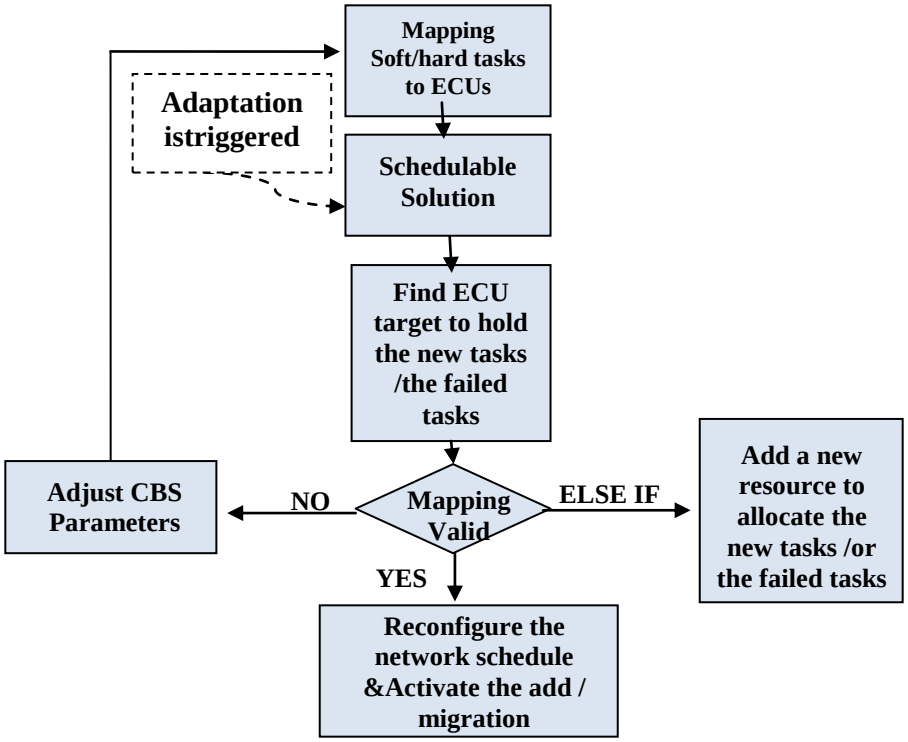

Figure4:Flowchart of tasks adaptation algorithm (i.e., TSeRBA)

Fig. 4 shows the flowchart of our algorithm TSeRBA (Tabu SearchReconfiguration and Bandwidth Allocation).The ASLA reconfiguration manager uses TSeRBA for adding and/or migrating applications at runtime. TSeRBA uses the following inputs: (1) a schedulable solution [2]obtained from the mapping manager and (2) the adaptation triggers from the monitor (for e.g., the request for adding a new application or a request for migrating a failed applications). The output of TSeRBA is a new system configuration which needs to be schedulable. To deal with the current adaptation, the algorithm starts by finding the target ECU to hold the new and /or the failed applications. TSeRBA maintains an ordered list of ECU candidates, andthe one which gives the best QoS is selected as a target ECU. This QoS is the probability of meeting the deadline for applications with soft real-time requirements and it depends on the allocated bandwidth Q_i[11]. Then if the mapping is still valid, the new application is simply added on the

[1]The task mapping algorithm is beyond the scope of this paper.
[2] A schedulable solution is the one that satisfies Liu & Layland utilization test for EDF [8].

target ECU (or the failed applications are simply migratedon the target ECU). Thus, in case of adding a new application the RM loads, instantiates and connects the new application, changes the network schedule and dynamically finds and binds to the correct interface. However, if the RM cannotanalyze this new application i.e., the mapping is not valid; in that case the bandwidth $Q_i$associated with applications with soft real-time requirements on that ECU is decreased, and then the new application can be mapped on the ECU. A completely new resource is added to hold the new application if the mapping is not valid even when we adjust the bandwidth.

Asla Plugins:
All applications will run on top of the ASLA plugins. ASLA plugins support the mechanisms for task reconfiguration and bandwidth allocation (i.e., TSeRBA algorithm) and also enable tasks to have guaranteed and protected access to required processing resources during reconfiguration in a timely manner.

ASLA DEVELOPMENT METHODOLOGY

The standard development process is slightly modified for introducing the necessary mechanisms for runtime adaptation.Yet, it remains fully compliant with the AUTOSAR development methodology [3]. There are additional elements that are introduced within this process for allowing the adaptation. The ASLA development process is depicted in Fig.5 and it consists of four steps:

The first step 'TheSystem Configuration' concerns the whole system. During this step the entire set of the applications is specified in terms of software architecture: SWCs, the Adaptive SWC, ports, real-time constraints, HW resource requirements and other information needed in the vehicle .In our approach we are interested in mixed critical applications with soft and hard real-time requirements.The output of this step is System Description –an AUTOSAR XML file, which serves as input for the following phase.

The second step 'Extract ASLAECU-Specific Information'concerns the SWCs and the Adaptive SWC implementation. (i.e. the definition of the internal behavior of the Runnables and the RTEvents). We consider that each SWC contains one runnable and is represented by one AUTOSAR task.During this phase the initial allocation of SWCs with both soft/hard real-time requirements to ECUs is specified and the application signals are mapped to bus frame.As a result of this step we obtain an initial solution (AUTOSAR XML)which is not necessary schedulable and itserves as input for the mapping manager and the following phase.

The third step 'ASLA ECUConfiguration 'concerns the configuration of the BSW modules and the RTE of each ECU. Our RTE extensioncontainsfunctions to support adaptation.The Adaptive SWC and its three components i.e., the RM, MM and the monitor are configured as tasks at the OS level with their corresponding runnables as application components. The callback functions configured at COM and the corresponding APIs defined at the RTE level are used for signal handling.This step includes the scheduling and the tasks mapping concepts. In our approach, we assume that the initial mapping of runnables to AUTOSAR tasks is given at design time similarly to AUTOSAR and all runnables are executed periodically within the context of an AUTOSAR Task. However, the mapping of tasks into the ECUs is performed using our task mapping algorithm O-TSMBA which we designed for the operational chain model. The output of O-

TSBMA algorithm is used as input for the runtime adaptation algorithm (i.e. TSeRBA) described in the previous section. Aresult of this step is theASLA ECU Configuration Description is generated aligned with the above steps. Finally, the software executablesare generated.

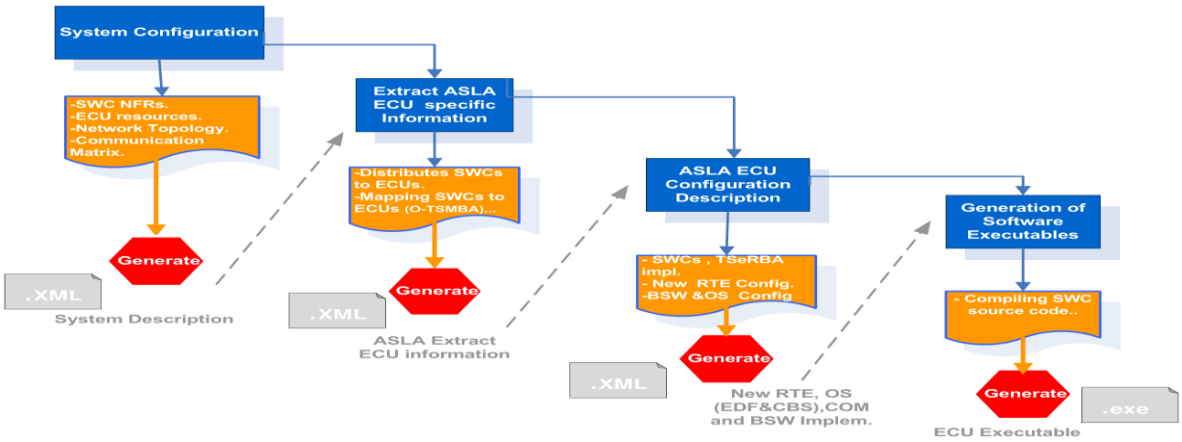

Figure 5:ASLA development process

OUR ONGOING IMPLEMENTATION OF ASLA

In order to create a rapid prototype of ALSA, we built an experimental platform (in Fig.8.).Three ARM-based STM32FDiscovery boards are used as representatives for more powerful control units that are expected in the future. The scenario we would like to show is described as follows: “upon the receipt of an adaptation trigger that is, one of the ECU fails, migrate tasks from thisECU to the other operational ECUs”. This section describes the software architecture in detail.

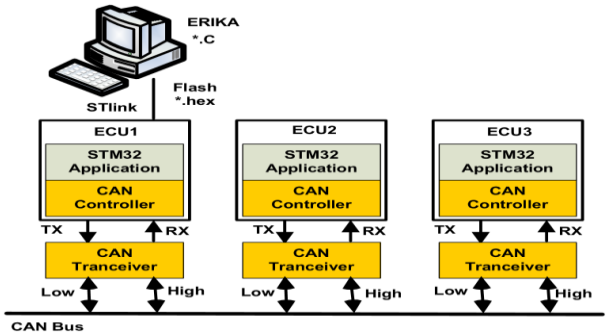

Figure 6: Block diagram of the experimental platform

HARDWARE

For the use case described above, we would at least need 3ECUs; one of the 3 ECUs acts as a system gateway and is connected to the PC using an USB cable. This ECU acts as the fault injection module and also to collect data from the system. The Fault in our case is “shutdown or restart of ECUs”. This fault and the collected data are controlled using the PC interface. Fig.6 shows the block diagram of our platform. For the demonstration purpose, we consider 3 applications with varying types of criticality namely, *Steer-By-Wire* (SBW) [12], *Wiper Status*

(WS) and *Door Locks (DL)*.

Steer-By-Wire Application. Enables the electric steering of the vehicle by sensing the driving and steering wheel angles, calculating the intended wheel angles, and actuating the change of direction via motors to the front axis.

Wiper Status Application.The wiper status application has three different modes: it can be off, constantly on or operated in an interval. The period of this interval depends on the amount of water on the windshield that is detected by a rain sensor.

Door Locks Application.Enables automatic locking and unlocking of doors in the vehicle. In case of an accident the door lock needs to be unlocked without a tangible delay; therefore, a fast wake of a sleeping door module is necessary. This application takes as input the Door state request with four valid states (Close, Open, Lock and Unlock). This function decides whether to accept the change depending on its current state.In our current implementation, each application is implemented on a separate ECU and it contains one runnable, considering several runnables is left as future work. The different applications and ASLA have been flashed using ERIKA enterprise [13]and STLINK as follows: ECU1 and ECU2 host WS,DL respectively. Whereas ECU3 executes SBW application as described in Table 1.

Runnable	Task	Bound ECU	CPU Utilization	CAN msg	Msg Lenght (bytes)	Bandwidth(Q)
SBW-Runnable	SBW-task	ECU3	0.32	SBW, ECU3-health-msg	2	-
WiperStatus-Runnable	WS-Task	ECU1	0.9	WS, ECU1-health-msg	2	58
DoorLock-Runnable	DL-Task	ECU2	0.49	DL, ECU2-health-msg	2	91

Table 1: System tasks and Allocation

SOFTWARE

The basic software running on the hardware includes the ERIKA-OS [13] operating system and generated code from RT-Druid[13].The code generated conforms to AUTOSAR4.x specifications and produce the minimum required implementation to produce a working system. An OIL file for configuration of ERIKA enterprise is also generated for each ECU. This generated code includes the necessary code modifications required as part of the runtime adaptation support described in the previous section. The GCC compiler for STM32Discovery board is used along with ERIKA OS configuration tool to produce an executable for each ECU.The CAN message identifiers aregenerated for each message on the bus. The various *Runnables, Tasks* and *Messages* used in the experimentalsystem are provided in Table1. The runnables parameters like WCET, Period and ECU were provided by the system designer. Message sizesand CAN configuration (125kps) were also specified at the design phase. Specific CAN identifiers are assigned to specific ECUs. For example, if the WS application is running on ECU1, the CAN identifier assigned to its message is 101…etc.

TEST PLAN AND PRELIMINARY RESULTS

In this section, we present how ASLA deals with the case of ECU failure through task-level adaptation as depicted in Figure 7.

Test Procedure

1. Switch on all the three ECUs.

2. Switch OFF ECU1 when one of the other operational ECUs has greater free CPU utilization.
3. Capture the logs.

Result

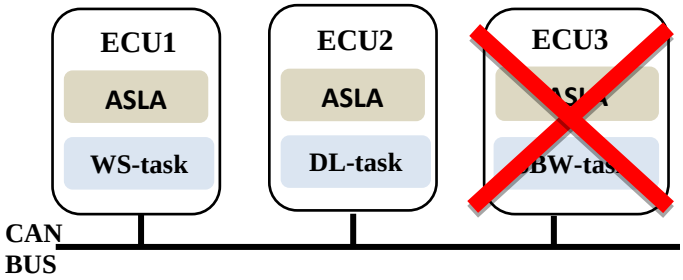

Figure7: Task migration example

Upon the receipt ofthe adaptation trigger 'ECU3 fails', the RM tries to find the target ECU to hold the failed tasks on ECU3 (i.e., SBW task) (as shown in Fig.7). ECU1 is determined as an infeasible solution since the combined utilization on the ECU1would exceed 100%if SBW-task would be allocated on ECU1 already hosting WS-task. Even by reducing the bandwidth of WS-task to zero, SBW-task would not meet its deadline (because the resulting task set on ECU1 is not schedulable). Hence, the RM updates the set of candidate ECUs and selects ECU2 as the target migration node, since the total CPU utilization available for DL-task on ECU2 if SBW-task is migrated on it is equal to (1-(0.49+0.32) =0.26. Therefore, the utilization desired by SBW-task can be made available by decreasing the bandwidth of the DL-task on ECU2. For instance, the available utilization and the changed bandwidth for DL-task are equal to $U_{DL-task} = 0.27$ and $Q^{new}_{DL-task} = 56$.

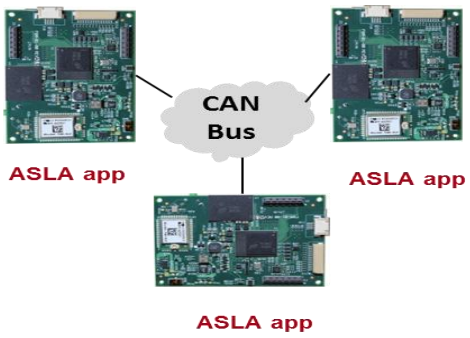

Figure 8 : Car electronics

CONCLUSION

Runtime adaptation of embedded systems was not a main concern in many safety critical application domains up to now. The economic pressure, for time-to market reasons, but also the new challenges posed by the Internet of Things (IoT), imposes a rapid evolution of embedded systems. Smart cities are example of a specific IoT where cars become connected objects. These new trends with the recent innovation concerning autonomous vehicles and ADAS (Advanced Driver Assistance Systems) raise the problem of evolution

of safety critical systems. This was one of the main motivations behind this paper: *Is it useful to make AUTOSAR adaptive?* Our work shows that AUTOSAR architecture as it is today does not offer enough flexibility to perform runtime adaptation and hence does not comply with the new trends discussed above. We have shown that with the new AUTOSAR architecture i.e., ASLA; runtime adaptation is however possible.This work opens up many directions for future research. Primarily, we intend to continue validating the results in the experimental platform, but also move it to an industrial setting and try it in a real vehicle. In addition, we envisage an extension to our approach to support mode change protocols for operational mode changes.

ACKNOWLEDGMENT

This research is partially supported by the EU FP7 project SafeAdapt.

REFRENCES

[1] AUTOSAR, http://www.autosar.org/.

[2] The OSEK/VDX portal. http://portal.osek-vdx.orgAccessed on 2013-11-12.

[3] AUTOSAR, http://www.autosar.org/specifications/release-42/methodology-and templates/AUTOSAR.

[4] AUTOSAR Partnership. Specification of the RTE level V2.0.1 R3.0 Rev 0001, February 2008.

[5] Saraswat Prabhat Kumar and Pop Paul and Madsen Jan. “Task Mapping and Bandwidth Reservation for Mixed Hard/Soft Fault-Tolerant Embedded Systems”[J].IEEE Computer Society, 2010, (978-0-7695-4001-6): 89—98.

[6] Davis, Robert I. and Burns, Alan and Bril, Reinder J. and Lukkien, Johan J.“Controller Area Network (CAN) Schedulability Analysis: Refuted, Revisited and Revised”[J]. Kluwer Academic Publishers, 2007,35(0922-6443):34.

[7] Abeni, L. and Buttazzo, G.“Integrating Multimedia Applications in Hard Real-Time Systems” [J], IEEE Computer Society, 1998, (0-8186-9212-X): 4.

[8] Liu, C. L. and Layland, James W.“Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment” [J], ACM, 1973, 20(0004-5411): 16.

[9] P. Pop, V. Izosimov, P. Eles and Z. Peng,“Design Optimization of Time- and Cost-Constrained Fault-Tolerant Embedded Systems With Checkpointing and Replication” [J], IEEE, 2009, 17(1063-8210): 3.

[10] Mitzlaff, Martin and Kapitza,Wolfgang. “Enabling Mode Changes in a Distributed Automotive System” [J]. ACM, 2010, (978-1-60558-915-2): 4.

[11] Abeni, Luca and Buttazzo, Giorgio C.“Stochastic Analysis of a Reservation Based System” [J]. IEEE Computer Society, 2001,(0-7695-0990-8): 92.

[12] Chaaban, Khaled and Leserf, Patrick and Saudrais, Sebastien. “Steer-by-wire System Development Using AUTOSAR Methodology” [J]. IEEE Press, 2009, (978-1-4244-2727-7): 8.

[13] Erika Enterprise web site, http://erika.tuxfamily.org/.